

# **Отчёт по лабораторной работе № 4**

**Дисциплина: Архитектура компьютера**

Румянцев Артём Олегович

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Задание</b>	<b>6</b>
<b>3</b>	<b>Теоретическое введение</b>	<b>7</b>
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>10</b>
4.1	Создание программы Hello world! . . . . .	10
4.2	Работа с транслятором NASM . . . . .	11
4.3	Работа с расширенным синтаксисом командной строки NASM . .	12
4.4	Работа с компоновщиком LD . . . . .	12
4.5	Запуск исполняемого файла . . . . .	13
4.6	Выполнение заданий для самостоятельной работы. . . . .	13
<b>5</b>	<b>Выводы</b>	<b>16</b>
<b>6</b>	<b>Список литературы</b>	<b>17</b>

## Список иллюстраций

4.1	Перемещение между директориями . . . . .	10
4.2	Создание пустого файла . . . . .	10
4.3	Открытие файла в текстовом редакторе . . . . .	10
4.4	Заполнение файла . . . . .	11
4.5	Компиляция текста программы . . . . .	11
4.6	Компиляция текста программы . . . . .	12
4.7	Передача объектного файла на обработку компоновщику . . . . .	12
4.8	Передача объектного файла на обработку компоновщику . . . . .	13
4.9	Запуск исполняемого файла . . . . .	13
4.10	Создание копии файла . . . . .	13
4.11	Изменение программы . . . . .	13
4.12	Компиляция текста программы . . . . .	14
4.13	Передача объектного файла на обработку компоновщику . . . . .	14
4.14	Запуск исполняемого файла . . . . .	14
4.15	Создании копии файлов в другом каталоге . . . . .	14
4.16	Добавление файлов на GitHub . . . . .	15
4.17	Отправка файлов . . . . .	15

## Список таблиц

# 1 Цель работы

Цель данной лабораторной работы - освоить процедуры компиляции и сборки программ, написанных на ассемблере NASM.

## 2 Задание

1. Создание программы Hello world!
2. Работа с транслятором NASM
3. Работа с расширенным синтаксисом командной строки NASM
4. Работа с компоновщиком LD
5. Запуск исполняемого файла
6. Выполнение заданий для самостоятельной работы.

### 3 Теоретическое введение

Основными функциональными элементами любой ЭВМ являются центральный процессор, память и периферийные устройства. Взаимодействие этих устройств осуществляется через общую шину, к которой они подключены. Физически шина представляет собой большое количество проводников, соединяющих устройства друг с другом. В современных компьютерах проводники выполнены в виде электропроводящих дорожек на материнской плате. Основной задачей процессора является обработка информации, а также организация координации всех узлов компьютера. В состав центрального процессора входят следующие устройства: - арифметико-логическое устройство (АЛУ) — выполняет логические и арифметические действия, необходимые для обработки информации, хранящейся в памяти; - устройство управления (УУ) — обеспечивает управление и контроль всех устройств компьютера; - регистры — сверхбыстрая оперативная память небольшого объёма, входящая в состав процессора, для временного хранения промежуточных результатов выполнения инструкций; регистры процессора делятся на два типа: регистры общего назначения и специальные регистры. Для того, чтобы писать программы на ассемблере, необходимо знать, какие регистры процессора существуют и как их можно использовать. Большинство команд в программах написанных на ассемблере используют регистры в качестве операндов. Практически все команды представляют собой преобразование данных хранящихся в регистрах процессора, это например пересылка данных между регистрами или между регистрами и памятью, преобразование (арифметические или логические

операции) данных хранящихся в регистрах. Доступ к регистрам осуществляется не по адресам, как к основной памяти, а по именам. Каждый регистр процессора архитектуры x86 имеет свое название, состоящее из 2 или 3 букв латинского алфавита. В качестве примера приведем названия основных регистров общего назначения (именно эти регистры чаще всего используются при написании программ): - RAX, RCX, RDX, RBX, RSI, RDI — 64-битные - EAX, ECX, EDX, EBX, ESI, EDI — 32-битные - AX, CX, DX, BX, SI, DI — 16-битные - AH, AL, CH, CL, DH, DL, BH, BL — 8-битные

Другим важным узлом ЭВМ является оперативное запоминающее устройство (ОЗУ). ОЗУ — это быстродействующее энергозависимое запоминающее устройство, которое напрямую взаимодействует с узлами процессора, предназначенное для хранения программ и данных, с которыми процессор непосредственно работает в текущий момент. ОЗУ состоит из одинаковых пронумерованных ячеек памяти. Номер ячейки памяти — это адрес хранящихся в ней данных. Периферийные устройства в составе ЭВМ: - устройства внешней памяти, которые предназначены для долговременного хранения больших объёмов данных. - устройства ввода-вывода, которые обеспечивают взаимодействие ЦП с внешней средой.

В основе вычислительного процесса ЭВМ лежит принцип программного управления. Это означает, что компьютер решает поставленную задачу как последовательность действий, записанных в виде программы.

Коды команд представляют собой многоразрядные двоичные комбинации из 0 и 1. В коде машинной команды можно выделить две части: операционную и адресную. В операционной части хранится код команды, которую необходимо выполнить. В адресной части хранятся данные или адреса данных, которые участвуют в выполнении данной операции. При выполнении каждой команды процессор выполняет определённую последовательность стандартных действий, которая называется командным циклом процессора. Он заключается в следующем: 1. формирование адреса в памяти очередной команды; 2. считывание кода команды из памяти и её дешифрация; 3. выполнение команды; 4. переход к



следующей команде.

Язык ассемблера (assembly language, сокращённо asm) — машинно-ориентированный язык низкого уровня. NASM — это открытый проект ассемблера, версии которого доступны под различные операционные системы и который позволяет получать объектные файлы для этих систем. В NASM используется Intel-синтаксис и поддерживаются инструкции x86-64.

## 4 Выполнение лабораторной работы

### 4.1 Создание программы Hello world!

С помощью утилиты `mkdir` создаю каталог, в котором буду работать и с помощью `cd` перехожу в него(рис. 1).

```
aorumyancev@Ubuntu:~$ mkdir -p ~/work/arch-pc/lab04  
aorumyancev@Ubuntu:~$ cd ~/work/arch-pc/lab04
```

Рис. 4.1: Перемещение между директориями

Создаю в текущем каталоге пустой текстовый файл `hello.asm` с помощью утилиты `touch` (рис. 2).

```
aorumyancev@Ubuntu:~/work/arch-pc/lab04$ touch hello.asm
```

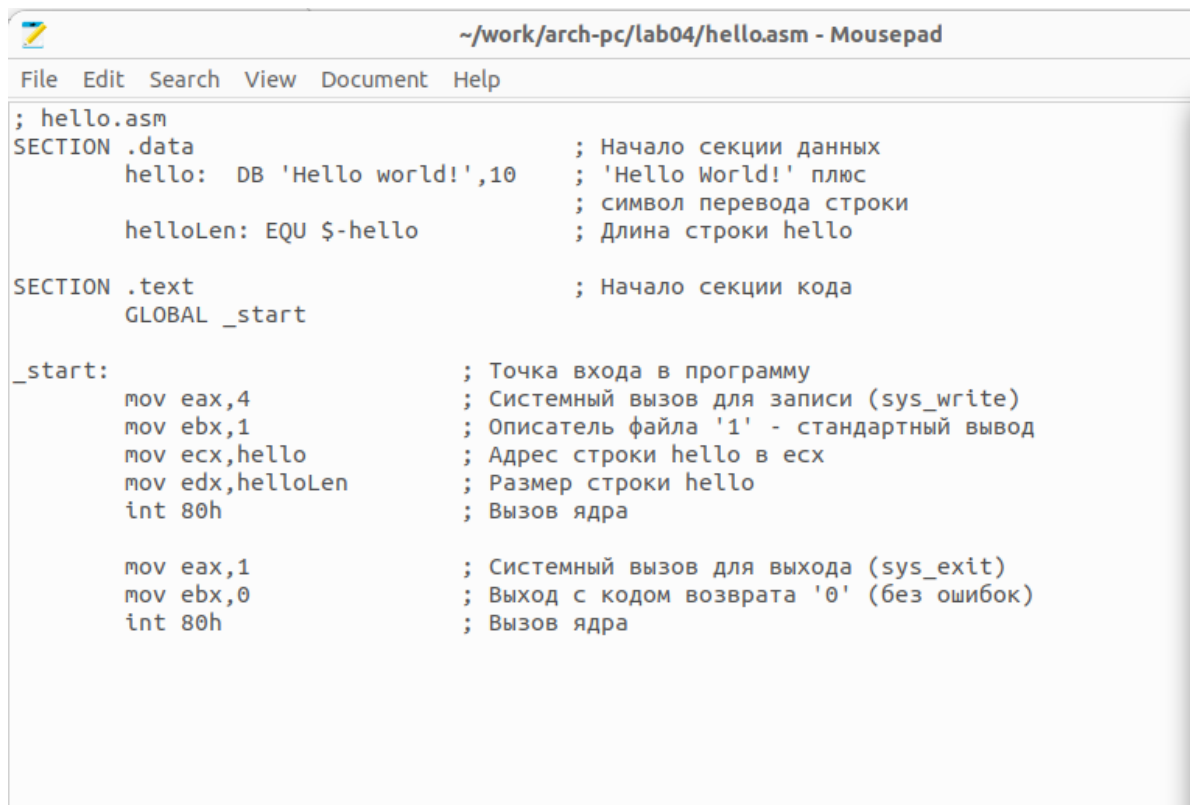
Рис. 4.2: Создание пустого файла

Открываю созданный файл в текстовом редакторе `mousepad` (рис. 3).

```
aorumyancev@Ubuntu:~/work/arch-pc/lab04$ mousepad hello.asm
```

Рис. 4.3: Открытие файла в текстовом редакторе

Заполняю файл, вставляя в него программу для вывода “Hello word!”. Так, как ассемблер не является высокоуровневым языком, каждая команда размещается на отдельной строке, так же обращаю внимание на регистр, так как Assembly чувствителен к нему. (рис. 4).



```
; hello.asm
SECTION .data                                ; Начало секции данных
    hello: DB 'Hello world!',10             ; 'Hello World!' плюс
                                           ; символ перевода строки
    helloLen: EQU $-hello                   ; Длина строки hello

SECTION .text                                ; Начало секции кода
    GLOBAL _start

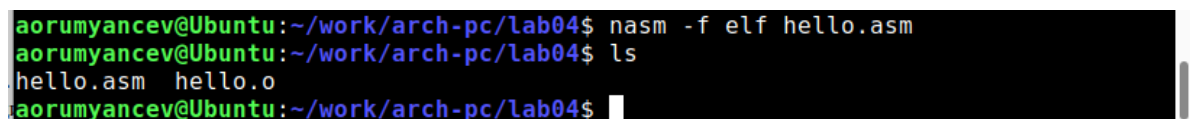
_start:                                      ; Точка входа в программу
    mov eax,4                               ; Системный вызов для записи (sys_write)
    mov ebx,1                               ; Описатель файла '1' - стандартный вывод
    mov ecx,hello                           ; Адрес строки hello в ecx
    mov edx,helloLen                        ; Размер строки hello
    int 80h                                 ; Вызов ядра

    mov eax,1                               ; Системный вызов для выхода (sys_exit)
    mov ebx,0                               ; Выход с кодом возврата '0' (без ошибок)
    int 80h                                 ; Вызов ядра
```

Рис. 4.4: Заполнение файла

## 4.2 Работа с транслятором NASM

Превращаю текст программы для вывода “Hello world!” в объектный код с помощью транслятора NASM, используя команду `nasm -f elf hello.asm`, ключ `-f` указывает транслятору `nasm`, что требуется создать бинарный файл в формате ELF (рис. 5). Далее проверяю правильность выполнения команды с помощью утилиты `ls`: действительно, создан файл “hello.o”.



```
aorummyancev@Ubuntu:~/work/arch-pc/lab04$ nasm -f elf hello.asm
aorummyancev@Ubuntu:~/work/arch-pc/lab04$ ls
hello.asm  hello.o
aorummyancev@Ubuntu:~/work/arch-pc/lab04$
```

Рис. 4.5: Компиляция текста программы

## 4.3 Работа с расширенным синтаксисом командной строки NASM

Ввожу команду, которая скомпилирует файл `hello.asm` в файл `obj.o`, используя ключ `-o` который задает имя объектному файлу, так же в файл будут включены символы для отладки (ключ `-g`), с помощью ключа `-l` будет создан файл листинга `list.lst` (рис. 6). Далее проверяю с помощью утилиты `ls` правильность выполнения команды.

```
aorumyancev@Ubuntu:~/work/arch-pc/lab04$ nasm -o obj.o -f elf -g -l list.lst hello.asm
aorumyancev@Ubuntu:~/work/arch-pc/lab04$ ls
hello.asm hello.o list.lst obj.o
aorumyancev@Ubuntu:~/work/arch-pc/lab04$
```

Рис. 4.6: Компиляция текста программы

## 4.4 Работа с компоновщиком LD

Передаю объектный файл `hello.o` на обработку компоновщику `LD`, чтобы получить исполняемый файл `hello` (рис. 7). Ключ `-o` задает имя создаваемого исполняемого файла. Далее проверяю с помощью утилиты `ls` правильность выполнения команды.

```
aorumyancev@Ubuntu:~/work/arch-pc/lab04$ ld -m elf_i386 hello.o -o hello
aorumyancev@Ubuntu:~/work/arch-pc/lab04$ ls
hello hello.asm hello.o list.lst obj.o
aorumyancev@Ubuntu:~/work/arch-pc/lab04$
```

Рис. 4.7: Передача объектного файла на обработку компоновщику

Выполняю следующую команду (рис. 8). Исполняемый файл будет иметь имя `main`, т.к. после ключа `-o` было задано значение `main`. Объектный файл, из которого собран этот исполняемый файл, имеет имя `obj.o`

```
aorumyancev@Ubuntu:~/work/arch-pc/lab04$ ld -m elf_i386 obj.o -o main
aorumyancev@Ubuntu:~/work/arch-pc/lab04$ ls
hello  hello.asm  hello.o  list.lst  main  obj.o
aorumyancev@Ubuntu:~/work/arch-pc/lab04$
```

Рис. 4.8: Передача объектного файла на обработку компоновщику

## 4.5 Запуск исполняемого файла

Запускаю на выполнение созданный исполняемый файл hello (рис. 9).

```
aorumyancev@Ubuntu:~/work/arch-pc/lab04$ ./hello
Hello world!
aorumyancev@Ubuntu:~/work/arch-pc/lab04$
```

Рис. 4.9: Запуск исполняемого файла

## 4.6 Выполнение заданий для самостоятельной работы.

С помощью утилиты cp создаю в текущем каталоге копию файла hello.asm с именем lab4.asm (рис. 10).

```
aorumyancev@Ubuntu:~/work/arch-pc/lab04$ cp hello.asm lab4.asm
aorumyancev@Ubuntu:~/work/arch-pc/lab04$
```

Рис. 4.10: Создание копии файла

С помощью текстового редактора mousepad открываю файл lab4.asm и вношу изменения в программу так, чтобы она выводила мои имя и фамилию. (рис. 11).

```
aorumyancev@Ubuntu:~/work/arch-pc/lab04$ mousepad lab4.asm
aorumyancev@Ubuntu:~/work/arch-pc/lab04$
```

Рис. 4.11: Изменение программы

Компилирую текст программы в объектный файл (рис. 12). Проверяю с помощью утилиты ls, что файл lab4.o создан.

```
aorumyancev@Ubuntu:~/work/arch-pc/lab04$ nasm -f elf lab4.asm
aorumyancev@Ubuntu:~/work/arch-pc/lab04$ ls
hello hello.asm hello.o lab4.asm lab4.o list.lst main obj.o
aorumyancev@Ubuntu:~/work/arch-pc/lab04$
```

Рис. 4.12: Компиляция текста программы

Передаю объектный файл lab4.o на обработку компоновщику LD, чтобы получить исполняемый файл lab4 (рис. 13).

```
aorumyancev@Ubuntu:~/work/arch-pc/lab04$ ld -m elf_i386 lab4.o -o lab4
aorumyancev@Ubuntu:~/work/arch-pc/lab04$ ls
hello hello.asm hello.o lab4 lab4.asm lab4.o list.lst main obj.o
aorumyancev@Ubuntu:~/work/arch-pc/lab04$
```

Рис. 4.13: Передача объектного файла на обработку компоновщику

Запускаю исполняемый файл lab4, на экран действительно выводятся мои фамилия и имя (рис. 14).

```
aorumyancev@Ubuntu:~/work/arch-pc/lab04$ ./lab4
Rumyancev Artem
aorumyancev@Ubuntu:~/work/arch-pc/lab04$
```

Рис. 4.14: Запуск исполняемого файла

Создаю копии файлов (рис.15)

```
aorumyancev@Ubuntu:~$ cp work/arch-pc/lab04/hello.asm work/study/2023-2024/"Архитектура компьютера"/arch-pc/labs/lab04
aorumyancev@Ubuntu:~$ cp work/arch-pc/lab04/lab4.asm work/study/2023-2024/"Архитектура компьютера"/arch-pc/labs/lab04
aorumyancev@Ubuntu:~$
```

Рис. 4.15: Создании копии файлов в другом каталоге

С помощью команд git add . и git commit добавляю файлы на GitHub.(рис. 17).

```

aorumyancev@Ubuntu:~/work/study/2023-2024/Архитектура компьютера/arch-pc$ git add .
aorumyancev@Ubuntu:~/work/study/2023-2024/Архитектура компьютера/arch-pc$ git commit -m "Add files for lab04"
[master 10ea635] Add files for lab04
 8 files changed, 290 insertions(+), 36 deletions(-)
 create mode 100644 labs/lab04/hello.asm
 create mode 100644 labs/lab04/lab4.asm
 rewrite labs/lab04/report/Makefile (83%)
 create mode 100644 labs/lab04/report/report.docx
 create mode 100644 labs/lab04/report/Л04_Румянцев_отчёт.docx
 create mode 100644 labs/lab04/report/Л04_Румянцев_отчёт.md
 create mode 100644 labs/lab04/report/Л04_Румянцев_отчёт.pdf
 create mode 100644 labs/lab4
aorumyancev@Ubuntu:~/work/study/2023-2024/Архитектура компьютера/arch-pc$

```

Рис. 4.16: Добавление файлов на GitHub

Отправляю файлы на сервер с помощью команды git push (рис. 17).

```

aorumyancev@Ubuntu:~/work/study/2023-2024/Архитектура компьютера/arch-pc$ git push
Enumerating objects: 17, done.
Counting objects: 100% (17/17), done.
Delta compression using up to 6 threads
Compressing objects: 100% (12/12), done.
Writing objects: 100% (12/12), 372.60 KiB | 2.74 MiB/s, done.
Total 12 (delta 3), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (3/3), completed with 2 local objects.
To github.com:aorumyancev/study_2023-2024_arch-pc.git
 61d6ba5..10ea635 master -> master
aorumyancev@Ubuntu:~/work/study/2023-2024/Архитектура компьютера/arch-pc$

```

Рис. 4.17: Отправка файлов

## **5 Выводы**

При выполнении данной лабораторной работы я освоил процедуры компиляции и сборки программ, написанных на ассемблере NASM.



## **6 Список литературы**

1. Архитектура компьютера. Лабораторная работа №4